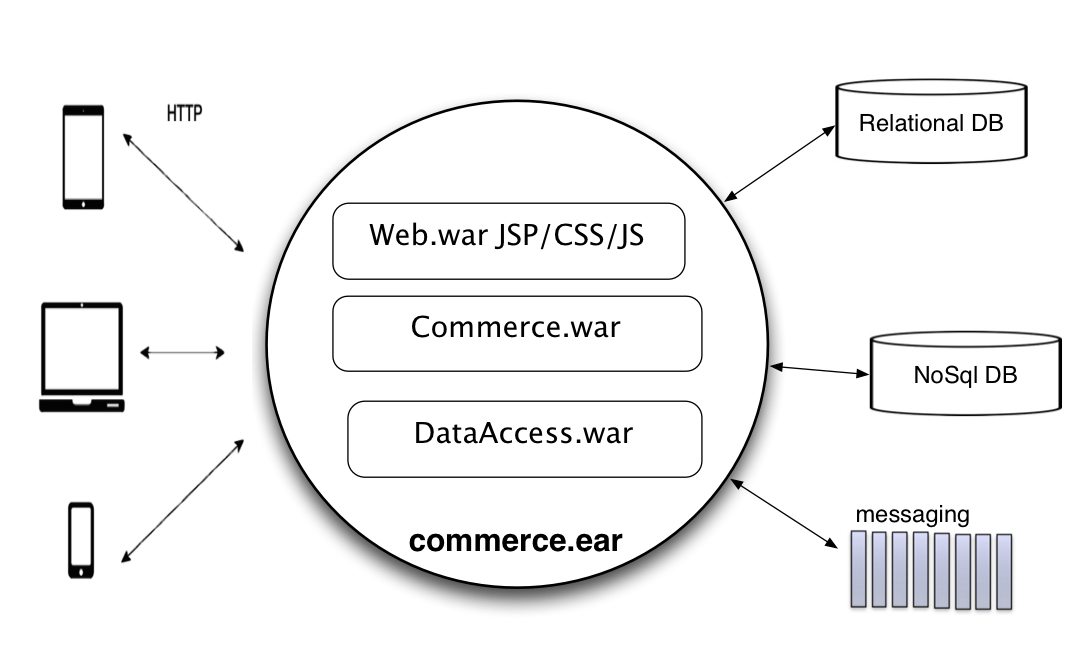
**Microservices Architecture-Retail**

Architectural choices are purely depending on technical and business needs, you need a solution when there is a problem or an opportunity, so the first thing is to work on identifying the problems and opportunities then set up short-term and long-term goals with statistical ROI measurements. Even today, there are few retailers who are using Websphere Commerce, SAP Hybris, Oracle ATG and Hybrid implementations and they are successful too, so, it is not a technology which makes organizations successful but, it helps to be successful.

However, in this article, I am going to talk about what is a microservices model and why most of the retailers are moving towards this architecture. Before that, let's take a look at monolith application architecture,

**Monolith Application Architecture:**

It is a single tiered application, where web interfaces and data interfaces are assembled in one application.

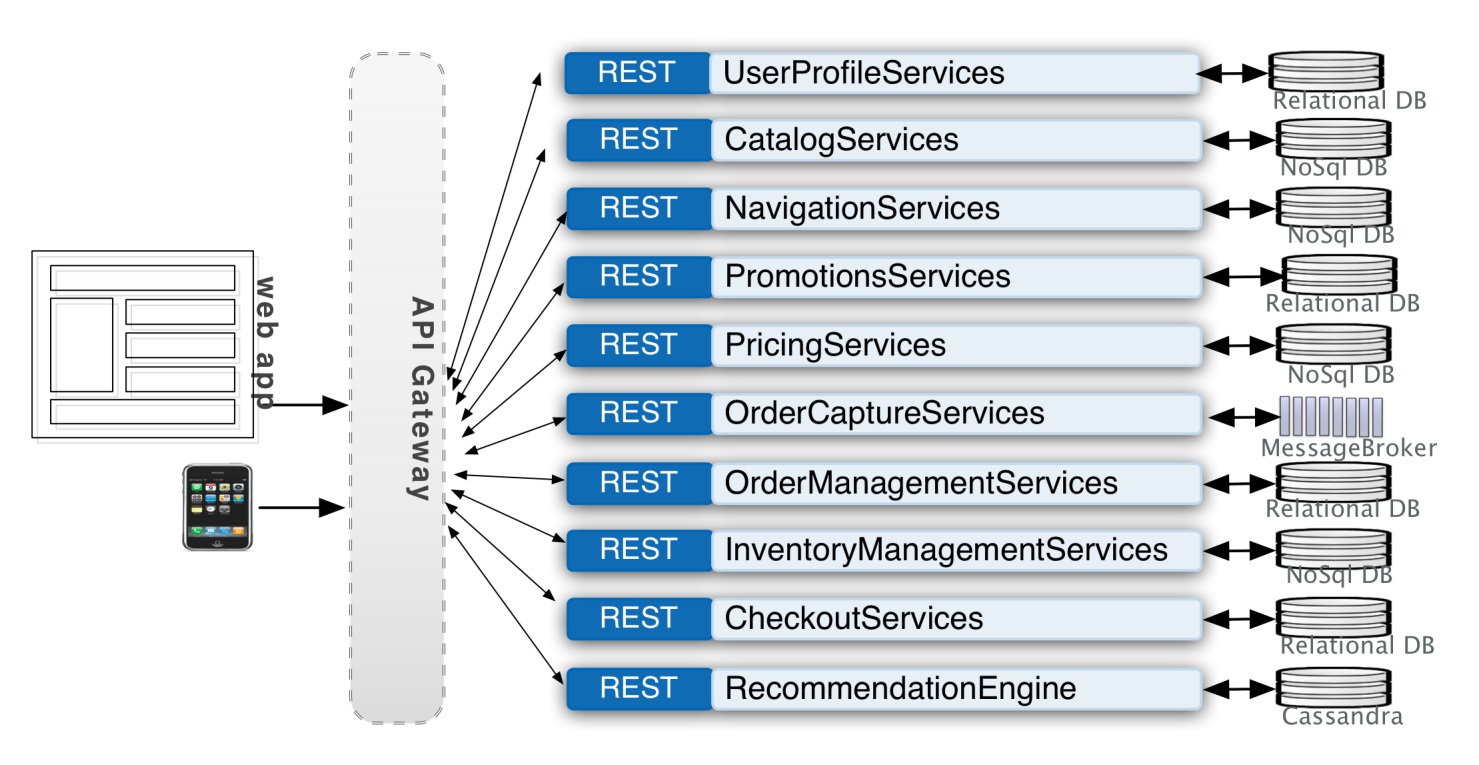


Some of the drawbacks with this architecture are,

1. Operational inefficiency: Every functional change requires to assemble, build, test and deploy entire application.
2. Longer release cycles: It is difficult to have shorter release cycles or feature releases, because, every other team is working on the same application. If you try to enforce shorter releases, it impacts the code quality and results with more defects.
3. No Ownership: There will be no ownership of a functional domain, everyone is working towards a single application, this approach leads to knowledge drain of core functionalities in commerce which ultimately results in delivery failures or delays.
4. Maintenance: Application is very large, and it is complex to understand and debug, also, maintenance of the application is going to be a nightmare.
5. No Fault tolerance: A defect like a memory leak in one function will bring down the complete application.

**Microservices Architecture:**

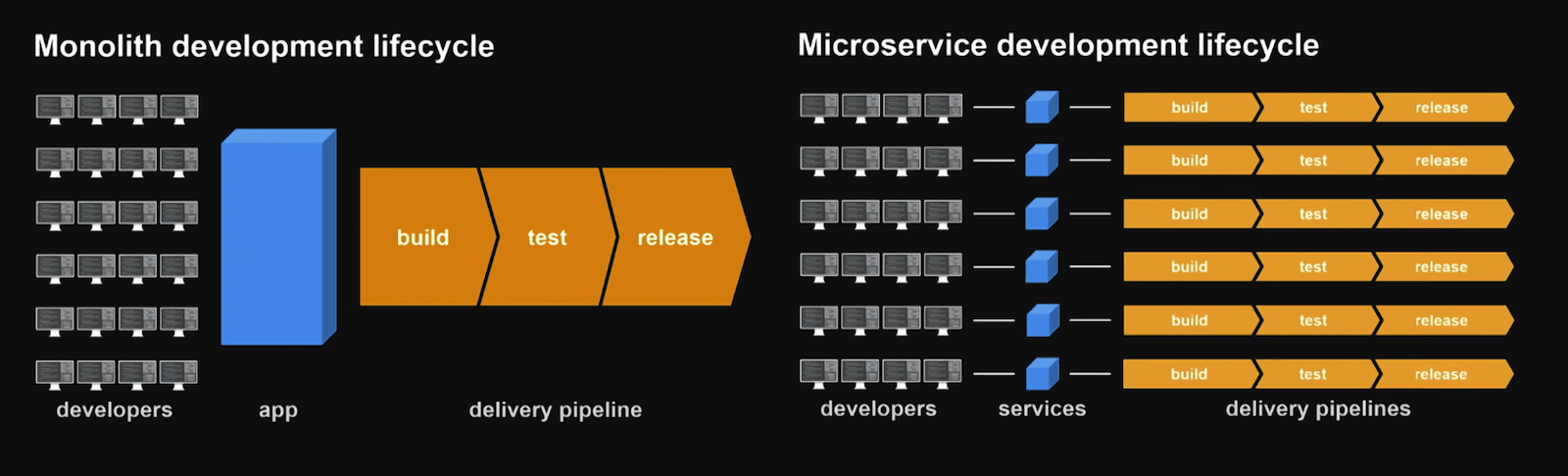
'Microservices', is an application architecture where smaller services/domains are developed, tested and deployed independently.



**Advantages**:

1. Ownership: It provides a clear ownership of each domain, which allows you to develop and release new functionalities within a given domain without depending on other development teams.
2. Fault isolation: If one service fails, it is still possible for the rest of the application to operate until that failed service recovered. Recovering a service is also not complex because it is an independent service and has its own build and deployment pipeline.
3. No need to bound to a single product or technology and you can save a lot from licensing, moreover you have the freedom to upgrade to new technologies and apply innovations.
4. High scalability: Based on the demand and performance these services can be deployed on a larger scale, in a monolith application you have to deploy same full one ear in every server which consumes more underlying resources.
5. Enabling Innovation: It is a key for long-term success, most of the companies are struggling to sustain because technology is evolving extremely fast and to be successful, it is required to innovate and beat the competition.
6. Business users get an opportunity to market and build their product strategies much quicker than compared monolith applications, where everything is interdependent.

**Development Lifecycle:**



**Challenges**:

Microservices architecture helps business to act quickly by delivering features with measured ROI's and time to market, however, there are some challenges with it,

1. It is highly depending on the DevOps, so, organizations need to adopt DevOps culture.
2. It is going to increase the operational complexity.
3. Interprocess communication is going to be challenging.
4. Serialization between services.

**Strategies to adopt Microservices:**

So far I have talked about what is a Microservices architecture, what are the benefits and challenges with it. Now, let's talk about how to adopt Microservices platform strategically. The first thing to understand is when and where it is required to apply microservices design pattern?

*'When the domain is customer facing and it is required to scale with the speed of innovation and time to market',*then, start with,

1. Decouple UI application from monolith ear.
2. Adopt DevOps and cloud-ready practices, identify the services eligible for the cloud-ready deployments.
3. Implement API Gateway, and API metrics dashboards to understand measurable metrics for APIs like, latency, SLA success rate, uptime and calls per day. This is where you will get an opportunity to measure technical ROI for your services investment.
4. Start looking at non-transactional domains which can be easily converted into a microservices model, for example: catalog browse and search functions.
5. Transactional domains: Before converting your bag, order and checkout domains into microservices, start with promotions, pricing, and inventory services, these are the backbones for order and checkout domains.
6. It is very important to build supporting components like caching, security, logging, monitoring, and reporting.
7. Leverage, non-relational distributed databases like Cassandra which provides high scalability and availability with great performance. It helps you to process and access large-scale data across retail systems, especially when you integrate physical stores with a digital store.
8. Leverage big data technologies like Kafka, Apache Storm, HBase, and Spark to feed real-time data to analytics dashboards, product rankings, search and product recommendation engines, this is where one can build and integrate next generation data driven platforms.
9. Finally, prefer asynchronous integrations by minimizing availability coupling.